EMBEDDED OPEN SOURCE SUMMIT

EMBEDDED LINUX CONFERENCE

Accelerated Porting of Linux, U-Boot and Yocto for Production-ready Embedded Systems

2024-04-17, Seattle

Vaishnav Achath

Keerthy J

Texas Instruments

About us: TI Processors and Open source

Decades of contribution and collaboration

Ingrained culture to give back to the community

Upstream FIRST!

Focus on long term, sustainable and quality products

Upstream and opensource ecosystem in device architecture

Upstream FIRST mentality!

Speakers | Intro

Vaishnav Achath, Software Engineer at Texas Instruments India. Vaishnav primarily works on Linux Kernel and U-Boot as part of the Texas Instruments Linux development team. Vaishnav is also a maintainer for TI platforms in Zephyr RTOS.

Keerthy J, SW Application Engineer with Texas Instruments India, as part of this role Keerthy primarily develops custom Linux/Boot loader features for customers & involved with custom board Linux bring-ups on TI TDA4 SOCs. He maintains TI bandgap & Davinci GPIO drivers.

Overview

- Motivation
- Challenges
- Bringing up a new SoC
  - Bringing up the Bootloader
  - Bringing up Linux
  - Adding Yocto support
- Bringing up a new hardware platform
  - U-Boot key care-abouts
  - Linux key care-abouts
- Case studies/Examples.
- Debugging Tools, tips and tricks.

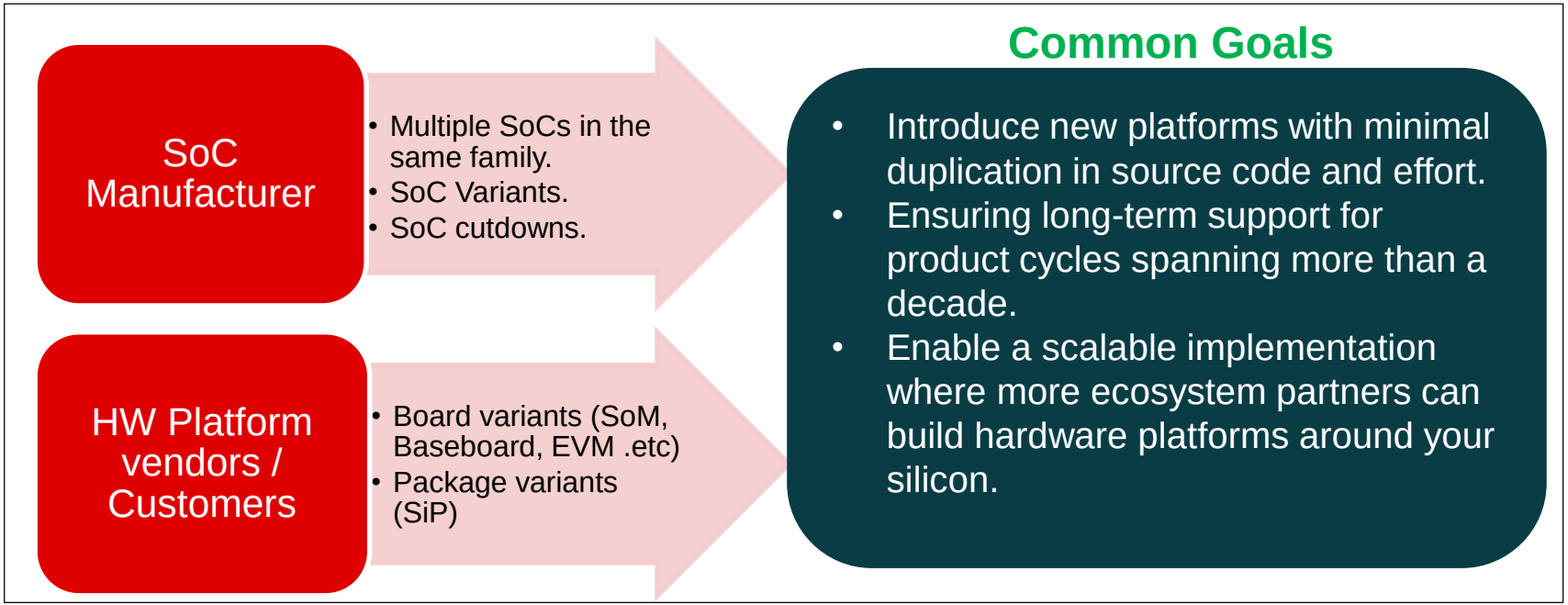
Most of the focus is diverted towards bringing up the bootloader as it is usually the most difficult part during new platform bringups.

Disclaimers

- This is a technology presentation, not product-readiness or roadmap commitment
- Opinions presented here are that of the speakers and may not reflect that of Texas Instruments Inc.
- The references and examples presented should be seen as a way to solve the challenges discussed, the speakers do not claim them as best practices.
- “Accelerated” porting in the title does not mean getting to the fastest U-Boot/Linux working port, sorry for the clickbait ;-) the intent is to discuss techniques and strategies that help reduce effort duplication when adding support for a new SoC or new hardware platform.

Motivation

Embarking on the journey of bringing up U-Boot, Linux, and Yocto on a freshly minted System on Chip or a new hardware platform can be a daunting task. The challenges are multiple folds and are usually much more than just bringing up these components in a system:



"Level of software support expected by customers today is more than just basic support during the product launch."

Challenges

TI AM62x SoC Family

SoC variants in family providing various levels of features, Example:

- Variants with different number of CPU cores.
- Variants with/without GPU.
- Variants with/without PRU.

TI AM62x Starter Kit

Single PCB Evaluation module with common features shared across multiple SoCs.

TI AM62x SIP

One of the above SoCs integrated into a System-In-Package with integrated PMIC, DDR, EEPROM .etc

TI AM68A Starter Kit

Multi-PCB (System on Module + Baseboard) Evaluation module.

TI DRA829 SoC

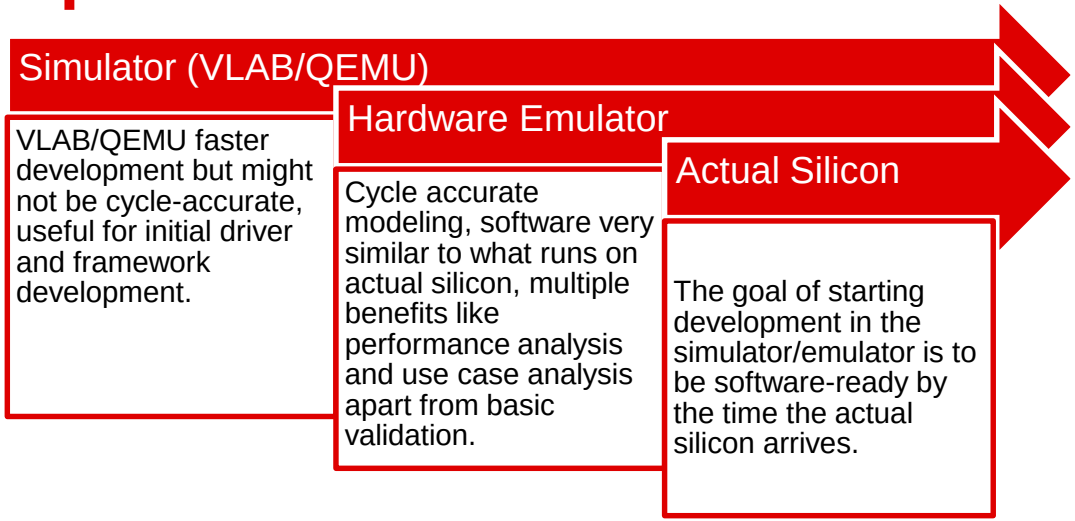
Same SoC, but different levels of security enforcement (GP, HS) due to differences in configuration.

TI J721S2 EVM

Multi-PCB (System on Module + Baseboard) Evaluation module with the same SoC variant

We have SoCs that look similar expose different capabilities, SoCs that looks similar go into boards that look different, SoCs that are exact same behave differently to software, how do we add support to all these combinations in a sustainable manner?

Bringing up a new SOC



Why perform software validation/ bring-up on a simulator/emulator?

- speed up actual silicon bring-up.
- Better debug mechanisms.
- validate silicon even before it is made, and fix critical bugs before actual hardware is made.
- Software readiness.

Reference: Accelerated Mainline Linux Development Ahead of SoC Availability

Bringing up the bootloader

If you reach till here, see console logs and DRAM is initialized, half of your bootloader bringup is complete. 😉

Step 1: Understand the boot sequence.

Example (armv7 SPL startup):

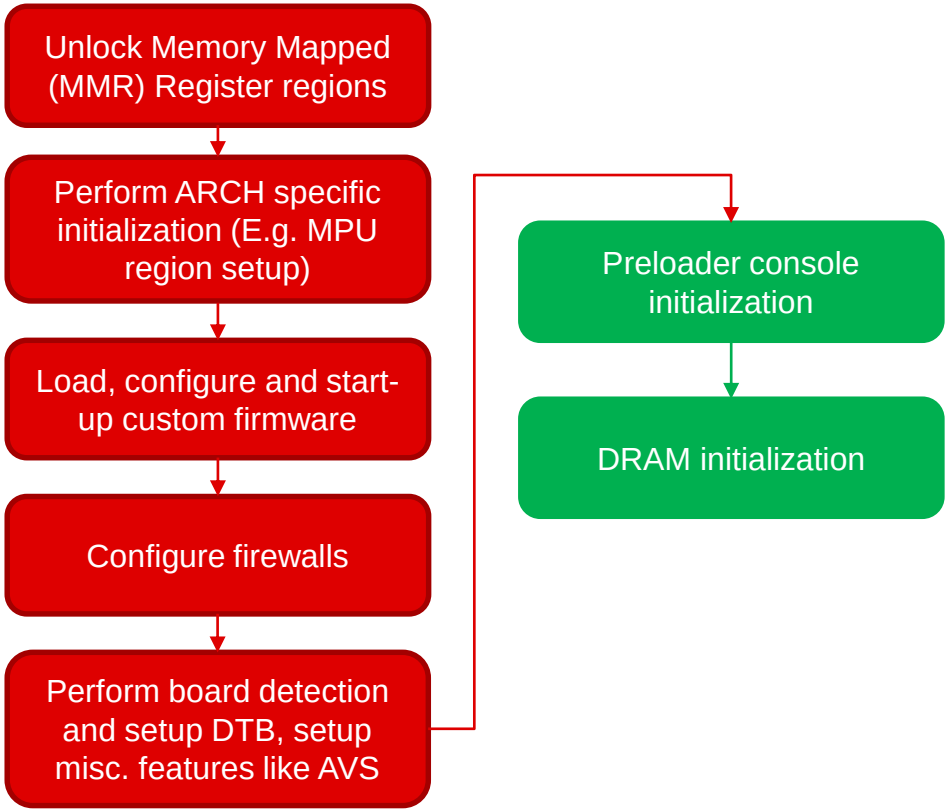
Architecture-specific (sometimes CPU-specific) initialization: essential init to reach board_init_f

- SDRAM initialization.
- Serial UART initialization (and anything that needs to make UART/SDRAM .etc work)
- global_data is available
- Stack is in SRAM
- BSS is not available _> no global/static variables

- main execution, common code.
- SDRAM, global_data, BSS is available.
- In SPL, stack and global_data is optionally relocated.
- For non-SPL, U-Boot is relocated to run at the top of memory.

How did we reach start.S?

How to implement a custom board_init_f()?



- board_init_f() just needs to do minimal things like initialize console, SDRAM.
- All the other extra functions implemented are either:
  - Dependencies to achieve the above.
  - Complexities arising due to multiple SoC/boards sharing same U-Boot build.
  - Miscellaneous features that needs to be enabled early in SPL.

How to customize the rest of the bootloader?

SPL override options

spl_boot_device() – reads hardware/ROM configuration and returns the bootmedia as per spl.h

spl_board_init() – board specific initialization in SPL, example PMIC specific initialization in SPL.

spl_perform_fixups() – Architecture/board specific fixups before processing boot payload (Example: enabling ECC for DDR, setting up DT for muxed boot peripherals).

dram_init_banksize() – implemented by boards to set DRAM bank information (Example: auto-detection of memory bank information or query device tree for memory bank information)

DTB fixup options

ft_system_setup() – Add system specific data into FDT before booting the OS (make sure to set CONFIG_OF_SYSTEM_SETUP). Example : Disable additional CPU cores, GPU .etc in a cutdown SoC version.

```
int ft_system_setup(void *blob, struct bd_info *bd)
{
    fdt_fixup_cores_nodes_am625(blob, k3_get_core_nr());
    fdt_fixup_gpu_nodes_am625(blob, k3_has_gpu());
    fdt_fixup_pru_node_am625(blob, k3_has_pru());
    fdt_fixup_thermal_zone_nodes_am625(blob, k3_get_max_temp());
    fdt_fixup_reserved(blob, "tfa", CONFIG_K3_ATF_LOAD_ADDR, 0x80000);
    fdt_fixup_reserved(blob, "optee", CONFIG_K3_OPTEE_LOAD_ADDR, 0x1800000);

    return 0;
}
```

ft_board_setup() – Add board-specific data to FDT before booting the OS. Example: setting up DT for muxed peripherals.

Bringing up Linux

Bringing up Linux on a custom platform is easier as it usually involves only creating a new device tree and bindings for the platform and adding minimal support in corresponding essential drivers, make sure to:

- Start with a minimal device tree with only basic peripherals.
- Use preloaded minimal intramfs for validation initial boot.
- Enable earlyconsole to enable early boot logs. (useful in case Serial initialization fails).
- Verify bootargs are proper for the init and console options.
- Implement SoC/JTAG ID detection if drivers depend on them.
- Check for drivers accepting SoC/board compatibles.
- If failures are seen with default configs, a minimal config can be used with features like RUNTIME_PM disabled and more debug features enabled.

arm64/dts/ti/Makefile:

```
# Boards with J722s SoC
dtb-$(CONFIG_ARCH_K3) += k3-j722s-evm.dtb
```

…/bindings/arm/ti/k3.yaml

```
    - description: K3 J722S SoC and Boards
      items:
        - enum:
            - ti,j722s-evm
        - const: ti,j722s
```

Bringing up Linux

drivers/dma/ti/k3-udma.c

```
static const struct soc_device_attribute k3_soc_devices[] = {
    { .family = "AM65X", .data = &am654_soc_data },
    { .family = "J721E", .data = &j721e_soc_data },
    { .family = "J7200", .data = &j7200_soc_data },
    { .family = "AM64X", .data = &am64_soc_data },
    { .family = "J721S2", .data = &j721e_soc_data},
    { .family = "AM62X", .data = &am64_soc_data },
    { .family = "AM62AX", .data = &am64_soc_data },
    { .family = "J784S4", .data = &j721e_soc_data },
    { .family = "AM62PX", .data = &am64_soc_data },
    { .family = "J722S", .data = &am64_soc_data },
    { /* sentinel */ }
};
```

drivers/soc/ti/k3-socinfo.c

```
static const struct k3_soc_id {
    unsigned int id;
    const char *family_name;
} k3_soc_ids[] = {
    { JTAG_ID_PARTNO_AM65X, "AM65X" },
    { JTAG_ID_PARTNO_J721E, "J721E" },
    { JTAG_ID_PARTNO_J7200, "J7200" },
    { JTAG_ID_PARTNO_AM64X, "AM64X" },
    { JTAG_ID_PARTNO_J721S2, "J721S2"},
    { JTAG_ID_PARTNO_AM62X, "AM62X" },
    { JTAG_ID_PARTNO_J784S4, "J784S4" },
    { JTAG_ID_PARTNO_AM62AX, "AM62AX" },
    { JTAG_ID_PARTNO_AM62PX, "AM62PX" },
    { JTAG_ID_PARTNO_J722S, "J722S" },
};
```

drivers/cpufreq/ti-cpufreq.c

```
static const struct of_device_id ti_cpufreq_of_match[] = {
    { .compatible = "ti,am33xx", .data = &am3x_soc_data, },
    { .compatible = "ti,am3517", .data = &am3517_soc_data, },
    { .compatible = "ti,am43", .data = &am4x_soc_data, },
    { .compatible = "ti,dra7", .data = &dra7_soc_data },
    { .compatible = "ti,omap34xx", .data = &omap34xx_soc_data, },
    { .compatible = "ti,omap36xx", .data = &omap36xx_soc_data, },
    { .compatible = "ti,am625", .data = &am625_soc_data, },
    { .compatible = "ti,am62a7", .data = &am625_soc_data, },
    { .compatible = "ti,am62p5", .data = &am625_soc_data, },
    /* legacy */
    { .compatible = "ti,omap3430", .data = &omap34xx_soc_data, },
    { .compatible = "ti,omap3630", .data = &omap36xx_soc_data, },
    {},
};
```

Be careful about:

- drivers using SoC identification to find configuration.
- drivers using the SoC compatible for match.

Adding a new machine configuration in Yocto

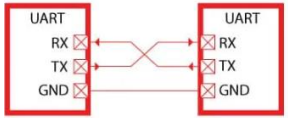
- Assuming there are already existing recipes and infrastructure for similar family devices, the following are the steps needed usually for a new device:
  - Create the machine configuration.
  - Specify overrides necessary for the platform, common overrides are:
    - Boot image files.
    - EXTRA_RRECOMMENDS - A list of recommended machine-specific packages to install as part of the image being built.
    - Firmware binaries
    - Update COMPATIBLE_MACHINE in necessary recipes.

Bringing up a new Hardware Platform

(Assuming the above steps are already done for the SoC)

U-Boot: Custom board bringup careabouts

- **Choice of boot media**
  - **SD** card is the best for bring ups. As it does not involve flashing using any external tools
  - **UART**: If memory based booting is not possible then DFU/UART are the preferred ones.
    - Most SoCs support the UART based boot mode.
    - It will be slower but it's the easiest to configure and test.



- **Board specific changes**: Even though the SOC is the same there are board specific components that need to taken care of while bringing up U-Boot/SPL on a custom board
  - **Pinmux**: Mainly take care of the pinmux for boot media, console & if needed - ethernet
  - **PMIC**: Enable the CONFIGS for the PMIC core & regulator drivers, DT nodes & hook the right power supplies from PMIC to SOC domains.
  - **DDR**: The most important function of SPL is to bring up the DDR. SPL DDR configurations mainly addresses the number of instances of DDR, the speed, total memory available, ECC enabling as per need.
  - **Console**: UART debug console prints are very important while bringing up custom board. So when a different UART is used as console, make sure pinmux, clock and baud are set correctly.

Linux custom board bring-up care abouts

- The first and foremost step is to enable earlycon to get the initial prints.
- Check for DDR stability in the U-Boot prompt using memtester.
- Carefully check the bootargs for the below parameters:
  - Console
  - Root
  - Add rootwait
- To eliminate unknowns just enable the basic peripherals needed for Linux boot i.e start with a minimal device tree:
  - Console UART
  - Rootfs hosting media
  - Timers
  - Linux booting cores
  - Pinmux for essential peripherals.
- Disable any other remote core loading in U-Boot:

Common Issues

- Some of the top issues & their potential root causes:
  - Crashing at random places with the same kernel image: Please check the DDR settings again this is most likely due to DDR instability.
  - Silent hangs without a single character on the console: Either the console param is wrong, or the clocks/pinmux for the console UART is wrong. If booting from U-Boot check the environment variables, if booting from kernel please add the bootargs in DT.
  - Some modules are enabled on SoC vendor platform may not be enabled on the custom board. Please evaluate the dts for such nodes and remove them or change their DT node status as disabled.
  - If some module is being probed & clock is on and not working as expected please examine their pin mux settings.
  - The most common hang/crash is seen when root is not properly set or set with wrong inputs:
    - By default SDK/EVM has SD card as rootfs.
    - Change that as needed to eMMC or UFS or RAMDISK.
  - UART RX is working but TX is not – Most likely the UART TX interrupt is not configured correctly validate against the TRM.

Case Studies/ Examples

What if you need to generate multiple types of images packaging different firmware components, handle different SoC variants etc.?

Binman helps you to manage packaging different FW components and generating U-Boot binaries with different configurations easily with a devicetree based configuration for each target:

- https://docs.u-boot.org/en/latest/develop/package/binman.html

```
#include "k3-j722s-binman.dtsi"
```

```
&binman {
    tiboot3-j722s-hs-evm.bin {
        filename = "tiboot3-j722s-hs-evm.bin";
        ti-secure-rom {
            content = <&u_boot_spl>, <&ti_fs_enc>, <&combined_tifs_cfg>,
                <&combined_dm_cfg>, <&sysfw_inner_cert>;
            combined;
            dm-data;
            sysfw-inner-cert;
            keyfile = "custMpk.pem";
            sw-rev = <1>;
            content-sbl = <&u_boot_spl>;
            content-sysfw = <&ti_fs_enc>;
```

Reference: Standardizing the generation and signing of boot images

Tired of maintaining and syncing a separate device tree for U-Boot and Linux?

with OF_UPSTREAM, the device tree needed is derived from <dts/upstream/src/..> making use of the device-tree rebasing repo:
git.kernel.org/pub/scm/linux/kernel/git/devicetree/devicetree-rebasing.git

instead of having duplicate DT sources which needed to be synced manually.

```
config OF_UPSTREAM
    bool "Enable use of devicetree imported from Linux kernel release"
    help
      Traditionally, U-Boot platforms used to have their custom devicetree
      files or copy devicetree files from Linux kernel which are hard to
      maintain and can usually get out-of-sync from Linux kernel. This
      option enables platforms to migrate to devicetree-rebasing repo where
      a regular sync will be maintained every major Linux kernel release
      cycle. However, platforms can still have some custom u-boot specific
      bits maintained as part of *-u-boot.dtsi files.

      Note: This option should be set in Kconfig, for the SoC as a whole.
      However, newer boards whose devicetree source files haven't landed in
      the dts/upstream subtree, they can override this option to have the
      DT build from existing U-Boot tree location instead.
```

Credits: Sumit Garg, Linaro Ltd. Reference: Rethinking U-Boot Devicetree Story

What if you have slight changes in configs for a configuration, Example: a different bootflow (say Android)?

```
$ make ARCH=arm am62x_evm_a53_defconfig
$ make ARCH=arm am62x_android_a53.config
```

```
CONFIG_USB_FUNCTION_FASTBOOT=y
CONFIG_FASTBOOT_BUF_ADDR=0xC0000000
CONFIG_FASTBOOT_BUF_SIZE=0x2F000000
CONFIG_FASTBOOT_FLASH=y
CONFIG_FASTBOOT_FLASH_MMC_DEV=0
CONFIG_FASTBOOT_CMD_OEM_FORMAT=y
# Enable Android boot flow
CONFIG_SYS_MALLOC_LEN=0x08000000
CONFIG_AVB_VERIFY=y
CONFIG_ANDROID_AB=y
CONFIG_ANDROID_BOOT_IMAGE=y
CONFIG_CMD_ADTIMG=y
CONFIG_CMD_ABOOTIMG=y
CONFIG_CMD_BCB=y
CONFIG_CMD_AB_SELECT=y
CONFIG_CMD_AVB=y
CONFIG_LIBAVB=y
# Store env in eMMC for dtbo overlays
CONFIG_ENV_SIZE=0x20000
CONFIG_ENV_OFFSET=0x100000
CONFIG_ENV_IS_IN_MMC=y
CONFIG_ENV_IS_NOWHERE=n
CONFIG_SYS_MMC_ENV_PART=1
CONFIG_SPL_ENV_IS_NOWHERE=y
```

defconfig fragments allow you to support extra features without bloating your default configurations.

What if you have multiple SoCs in a family with different variants providing different capabilities but the base feature set are same?

- ft_system_setup() helps you to override the device tree seen by OS, this is useful when the platforms are indifferent to U-Boot but the differences are only important for Linux.
- Fixups can be based on hardware register values (Example: Device Feature Registers, JTAG ID.etc)

```
int ft_system_setup(void *blob, struct bd_info *bd)
{
    fdt_fixup_cores_nodes_am625(blob, k3_get_core_nr());
    fdt_fixup_gpu_nodes_am625(blob, k3_has_gpu());
    fdt_fixup_pru_node_am625(blob, k3_has_pru());
    fdt_fixup_thermal_zone_nodes_am625(blob, k3_get_max_temp());
    fdt_fixup_reserved(blob, "tfa", CONFIG_K3_ATF_LOAD_ADDR, 0x80000);
    fdt_fixup_reserved(blob, "optee", CONFIG_K3_OPTEE_LOAD_ADDR, 0x1800000);

    return 0;
}
```

```
static inline int k3_get_core_nr(void)
{
    u32 full_devid = readl(CTRLMMR_WKUP_JTAG_DEVICE_ID);

    return (full_devid & JTAG_DEV_CORE_NR_MASK) >> JTAG_DEV_CORE_NR_SHIFT;
}
```

```
static inline int k3_has_pru(void)
{
    u32 full_devid = readl(CTRLMMR_WKUP_JTAG_DEVICE_ID);
    u32 feature_mask = (full_devid & JTAG_DEV_FEATURES_MASK) >>
                       JTAG_DEV_FEATURES_SHIFT;

    return !(feature_mask & JTAG_DEV_FEATURE_NO_PRU);
}
```

SoC variant with additional features

- If the memory map is consistent across the devices and if one device is a superset/subset of other, it helps to include one of the SoC dtsi and just perform the overrides in the superset/subset device.

```
// SPDX-License-Identifier: GPL-2.0-only OR MIT
/*
 * Device Tree Source for J722S SoC Family
 *
 * Copyright (C) 2024 Texas Instruments Incorporated - https://www.ti.com/
 */

#include <dt-bindings/gpio/gpio.h>
#include <dt-bindings/interrupt-controller/irq.h>
#include <dt-bindings/interrupt-controller/arm-gic.h>
#include <dt-bindings/soc/ti,sci_pm_domain.h>

#include "k3-am62p5.dtsi"
```

Include superset/subset device dtsi

```
/* Main domain overrides */

&inta_main_dmss {
    ti,interrupt-ranges = <7 71 21>;
};

&oc_sram {
    reg = <0x00 0x70000000 0x00 0x40000>;
    ranges = <0x00 0x00 0x70000000 0x40000>;
};
```

Then perform overrides in the device tree.

Debugging Tools, Tips and Tricks

This repo contains a list and usage examples of commonly used u-boot commands and Linux utilities that help during bringups:

https://github.com/vaishnavachath/boardbringup-swissknife

What if you don’t even reach to U-boot console and have no prints to debug?

Step 1: Don’t panic

Step 2: You can panic a little bit.

Step 3: Use OpenOCD to debug with JTAG

Debugging Heterogenous SoC using OpenOCD

Summary

- Starting the bringup on a Simulator/Emulator if available is very helpful to validate the software and ensure software readiness before silicon.
- Always start from the latest upstream baseline and continue development on this baseline, this helps to upstream your changes much easier.
- Whenever you have a working version ready, remember to send **UPSTREAM FIRST**, it helps to get early feedback from the community and it is much easier to fix your implementation according to community expectations before shipping out to customers. More complex the feature is, more important it is to get upstream feedback first.
- For Linux Device tree changes, make sure to strictly follow DTS coding guidelines (Documentation/devicetree/bindings/dts-coding-style.rst) and also perform dt_binding_check on new bindings and dtbs_check on new device tree added.
- It is more efficient to start with the Linux DT from the upstream baseline first and then derive the U-Boot device tree from Linux instead of having a separate device tree for U-Boot during bringup.
- Be active in the community, so that you don’t miss out on new features and best practices like Binman, OF_UPSTREAM .etc.

References

- Standardizing the generation and signing of boot images
- Debugging Heterogenous SoC using OpenOCD
- Rethinking U-Boot Devicetree Story
- Porting U-Boot and Linux on new ARM boards: a step-by step guide

Credits and Acknowledgement

Thank you!

- Texas Instruments Inc.
- The Linux Foundation.
- Vignesh Raghavendra, Texas Instruments
- Nishanth Menon, Texas Instruments
- Andrew F Davis, Texas Instruments
- Neha Francis, Texas Instruments
- Bryan Bratloff, Texas Instruments
- Sumit Garg, Linaro Ltd.

Q&A

- Contact Information:
  - Vaishnav Achath <vaishnav.a@ti.com>
  - Keerthy J <keerthy@ti.com>
- Also on IRC @ libera.chat #linux-ti

Learn more about TI products

- https://www.ti.com/linux
- https://www.ti.com/processors
- https://www.ti.com/edgeai

Why choose TI MCUs and processors?

Scalability

Our products offer scalable performance that can adapt and grow as the needs of your customers evolve.

Efficiency

We design products that extend battery life, maximize performance for every watt expended, and unlock the highest levels of system efficiency.

Affordability

We strive to make innovation accessible to all by creating cost-effective products that feature state-of-the-art technology and package designs.

Availability

Our investment in internal manufacturing capacity provides greater assurance of supply, supporting your growth for decades to come.